Sparse Matrix Factorization: Applications to Latent Semantic Indexing

Erin Moulding, Department of Computer Science,
University of Saskatchewan, Saskatoon, SK, S7N 5C9, erm764@mail.usask.ca

April Kontostathis, Department of Mathematics and Computer Science,
Ursinus College, Collegeville PA 19426, akontostathis@ursinus.edu

Raymond J. Spiteri, Department of Computer Science,
University of Saskatchewan, Saskatoon, SK, S7N 5C9, spiteri@cs.usask.ca

---

Abstract—This article describes the use of Latent Semantic Indexing (LSI) and some of its variants for the TREC Legal batch task. Both folding-in and Essential Dimensions of LSI (EDLSI) appeared as if they might be successful for recall-focused retrieval on a collection of this size. Furthermore, we developed a new LSI technique, one which replaces the Singular Value Decomposition (SVD) with another technique for matrix factorization, the sparse column-row approximation (SCRA). We were able to conclude that all three LSI techniques have similar performance. Although our 2009 results showed significant improvement when compared to our 2008 results, the use of a better method for selection of the parameter K, which is the ranking that results in the best balance between precision and recall, appears to have provided the most benefit.

1 INTRODUCTION

Our submissions for the 2007 and 2008 TREC Legal competitions tested a variety of both simple and state-of-the-art normalization systems. For the 2009 TREC Legal Competition, we tested a new matrix factorization for Latent Semantic Indexing (LSI).

We have extensive experience with LSI, including previous years in TREC legal competitions. We also have used the new Essential Dimensions of LSI (EDLSI) approach, and tested it on small and large datasets. EDLSI allows a much smaller LSI dimensionality reduction parameter (k) to be used, and weights the LSI results with the results from traditional vector-space retrieval. EDLSI has been shown to consistently match or improve on LSI [8]. This suggested to us that the PSVD approximation used in LSI may not be ideal, and instead, an approximation should be used that somehow captures more of the term-document matrix. Investigating this led to the sparse column-row approximation (SCRA), which

we test here in its first application to information retrieval with datasets of this order of magnitude.

Teams this year were permitted only three runs. Our first run was a LSI run using folding-in, the second was a fully distributed EDLSI run, and the third was a fully distributed run using the SCRA. As in last year's competition, teams were required to set K and K_h for each query. These parameters indicate where the system believes that precision and recall are best balanced for relevant and highly relevant documents respectively. The balance is indicated by the scoring measure $F1@K$:

$$F1@K = \frac{2 * P@K * R@K}{P@K + R@K}$$

The rest of this paper is organized as follows. Section 2 presents necessary background information on the methods used. Section 3 describes our approach and the different runs submitted. Section 4 presents the results, and Section 5 gives our conclusions.

2 BACKGROUND

In this section we begin with a description of vector-space retrieval, which forms the foundation for Latent Semantic Indexing (LSI). We also present a brief overview of LSI [3], including the technique of folding-in. We discuss Essential Dimensions of LSI (EDLSI) and its improvements over LSI. We also describe the sparse column-row approximation (SCRA) used for our new work.

2.1 Vector-Space Retrieval

In vector-space retrieval, a document is represented as a vector in t-dimensional space, where t is the number

Report Documentation Page		

1. REPORT DATE **NOV 2009**	2. REPORT TYPE	3. DATES COVERED **00-00-2009 to 00-00-2009**

4. TITLE AND SUBTITLE **Sparse Matrix Factorization: Applications to Latent Semantic Indexing**	5a. CONTRACT NUMBER
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Ursinus College,Department of Mathematics and Computer Science,Collegeville,PA,19426**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES
Proceedings of the Eighteenth Text REtrieval Conference (TREC 2009) held in Gaithersburg, Maryland, November 17-20, 2009. The conference was co-sponsored by the National Institute of Standards and Technology (NIST) the Defense Advanced Research Projects Agency (DARPA) and the Advanced Research and Development Activity (ARDA).

14. ABSTRACT
see report

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**Same as Report (SAR)**	**8**	

of terms in the lexicon being used. If there are d documents in the collection, then the vectors representing the documents can be represented by a matrix $A \in \Re^{t \times d}$, called the *term-document matrix*. Entry $a_{i,j}$ of matrix A indicates how important term i is to document j, where $1 \le i \le t$ and $1 \le j \le d$.

The entries in A can be binary numbers (1 if the term appears in the document and 0 otherwise), raw term frequencies (the number of times the term appears in the document), or weighted term frequencies. Weighting can be done using either local weighting, global weighting, or a combination of both. The purpose of local weighting is to capture the relative importance of a term within a specific document; therefore, local weighting uses the frequency of the term within the document to calculate the weight and assigns a higher weight if the frequency is higher. The purpose of global weighting is to identify terms that discriminate effectively between documents; thus, global weighting uses the frequency of the term within the entire document collection to calculate the weight and assigns a higher weight if the frequency is lower. Because document size often varies widely, the weights are also usually normalized; otherwise, long documents are more likely to be retrieved. See, e.g., [1], [12] for a comprehensive discussion of local and global weighting techniques.

Common words, such as *and*, *the*, *if*, etc., are considered to be *stop-words* [1] and are not included in the term-document matrix. Words that appear infrequently are often excluded to reduce the size of the lexicon.

Like the documents, queries are represented as t-dimensional vectors, and the same weighting is applied to them. Documents are retrieved by mapping $\mathbf{q}$ into the row (document) space of the term-document matrix, A:

$$\mathbf{w} = \mathbf{q}^T A.$$

After this calculation, $\mathbf{w}$ is a d-dimensional row vector, entry j of which is a measure of how relevant document j is to query $\mathbf{q}$. In a traditional search-and-retrieval application, documents are sorted based on their relevance score (i.e., vector $\mathbf{w}$) and returned to the user with the highest-scoring document appearing first. The order in which a document is retrieved is referred to as the *rank*[1] of the document with respect to the query. The experiments in this paper run multiple queries against a given dataset, so in general the query vectors $\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_n$ are collected into a matrix $Q \in \Re^{t \times n}$ and their relevance scores are computed as

$$W = Q^T A,$$

where entry $w_{j,k}$ in $W \in \Re^{n \times d}$ is a measure of how relevant document j is to query k.

---

1. This rank is unrelated to the *rank* of a matrix mentioned below.

There are two immediate deficiencies of vector-space retrieval. First, W might pick up documents that are not relevant to the queries in Q but contain some of the same words. Second, Q may overlook documents that are relevant but that do not use the exact words being queried. The *partial singular value decomposition* (PSVD) that forms the heart of LSI is used to capture term relationship information in the term-document space. Documents that contain relevant terms but perhaps not exact matches will ideally still end up 'close' to the query in the LSI space [3].

2.2 Latent Semantic Indexing

LSI uses the PSVD to approximate A, alleviating the deficiencies of vector-space retrieval described above.

The PSVD, also known as the truncated SVD, is derived from the SVD. The (reduced) SVD decomposes the term-document matrix into the product of three matrices: $U \in \Re^{t \times r}$, $\Sigma \in \Re^{r \times r}$, and $V \in \Re^{d \times r}$, where r is the rank of the matrix A. The columns of U and V are orthonormal, and Σ is a diagonal matrix, the diagonal entries of which are the r non-zero singular values of A, customarily arranged in non-increasing order. Thus A is factored as

$$A = U\Sigma V^T.$$

The PSVD produces an optimal rank-k ($k < r$) approximation to A by truncating Σ after the first (and largest) k singular values. The corresponding columns from $k+1$ to r of U and V are also truncated, leading to matrices $U_k \in \Re^{t \times k}$, $\Sigma_k \in \Re^{k \times k}$, and $V_k \in \Re^{d \times k}$. A is then approximated by

$$A \approx A_k = U_k \Sigma_k V_k^T.$$

In the context of LSI, there is evidence to show that A_k provides a better model of the semantic structure of the corpus than the original term-document matrix was able to provide for some collections [3], [4], [5], [2]. For example, searchers may choose a term, t_1, that is synonymous with a term, t_2, that appears in a given document, d_1. If k is chosen appropriately and there is ample use of the terms t_1 and t_2 in other documents (in similar contexts), the PSVD will give t_1 a large weight in the d_1 dimension of A_k even though t_1 does not appear in d_1. Similarly, an ancillary term t_3 that appears in d_1, even though d_1 is not 'about' t_3, may well receive a lower or negative weight in A_k matrix entry (t_3, d_1).

Choosing an optimal LSI dimension k for each collection remains elusive. Traditionally, an acceptable k has been chosen by running a set of queries with known relevant document sets for multiple values of k. The k that results in the best retrieval performance is chosen as

the optimal k for each collection. Optimal k values are typically in the range of 100–300 dimensions [4], [10].

2.3 Folding-In

The PSVD is useful in information retrieval, but calculating it is computationally expensive [16]. This expense is offset by the fact that the LSI space for a given set of documents can be used for many queries. However, if terms or documents are added to the initial dataset, then either the PSVD must be recomputed for the new A or the new information must be added to the current PSVD. In LSI, the two traditional ways of adding information to an existing PSVD are folding-in [11], [2] and updating [16]. We describe folding-in here.

Folding-in is computationally inexpensive, but the performance of LSI generally deteriorates as documents are folded-in [2], [11]. In order to fold-in new document vectors, the documents are first projected into the reduced k-dimensional LSI space and then appended to the bottom of V_k. Suppose $D \in \Re^{t \times p}$ is a matrix whose columns are the p new documents to be added. The projection is

$$D_k = D^T U_k \Sigma_k^{-1},$$

and D_k is then appended to the bottom of V_k, so the PSVD is now

$$[\ A, \quad D\ ] \approx U_k\,\Sigma_k\,[\ V_k^T, \quad D_k^T\ ].$$

Note that U_k and Σ_k are not modified when using folding-in. A similar procedure can be applied to U_k to fold-in terms [2]. Folding-in may be useful if only a few documents are to be added to the dataset or if word usage patterns do not vary significantly, as may be the case in a restricted lexicon. However, in general, each application of folding-in corrupts the orthogonality of the columns of U_k or V_k, eventually reducing the effectiveness of LSI. It is not generally advisable to rely exclusively on folding-in when the dataset changes frequently [15].

2.4 Essential Dimensions of LSI

As the dimension of the LSI space k approaches the rank r of the term-document matrix A, LSI approaches vector-space retrieval. In particular, vector-space retrieval is equivalent to LSI when $k = r$.

Figures 1–3 show graphically that the performance of LSI may essentially match (or even exceed) that of vector-space retrieval even when $k \ll r$. For the CACM [13] and NPL [6] collections, we see that LSI retrieval performance continues to increase as additional dimensions are added, whereas retrieval performance of LSI for the MED collection peaks when $k = 75$ and then decays to the level of vector-space retrieval. Thus

we see that vector-space retrieval outperforms LSI on some collections, even for relatively large values of k. Other examples of collections that do not benefit from LSI can be found in [9] and [7].

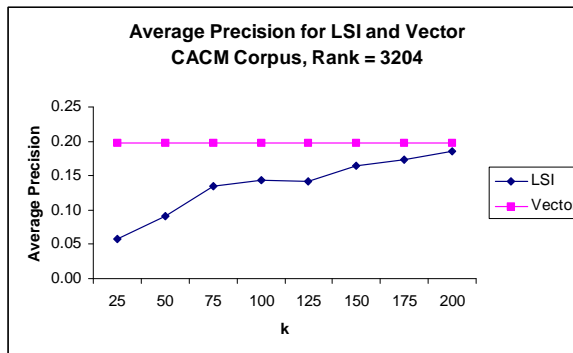Fig. 1. LSI vs. vector-space retrieval for the CACM Corpus ($r = 3\,204$).

Fig. 2. LSI vs. vector-space retrieval for the NPL Corpus ($r = 6\,988$).
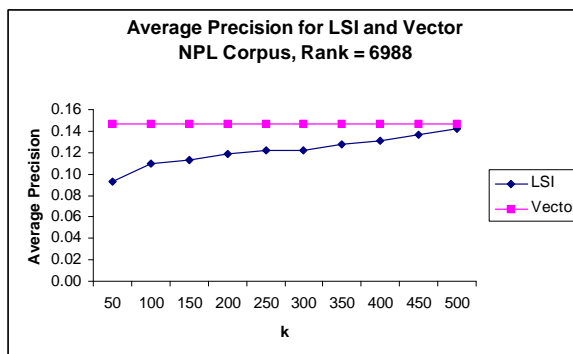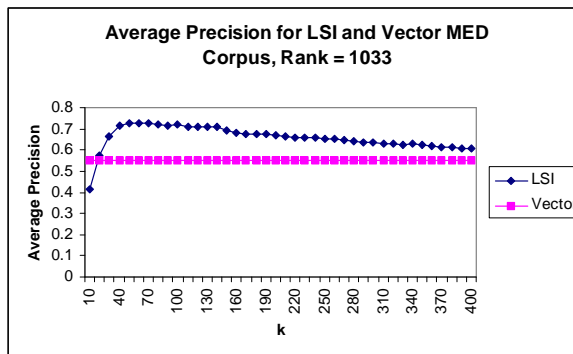

Fig. 3. LSI vs. vector-space retrieval for the MED Corpus ($r = 1\,033$).

These data suggest that we can use the term relationship information captured in the first few SVD vectors, in combination with vector-space retrieval, a technique referred to as Essential Dimensions of Latent

Semantic Indexing (EDLSI). Kontostathis demonstrated good performance on a variety of collections by using only the first 10 dimensions of the SVD [8]. The model obtains final document scores by computing a weighted average of the traditional LSI score using a small value for k and the vector-space retrieval score. The result vector computation is

$$W = x(Q^T A_k) + (1-x)(Q^T A),$$

where x is a weighting factor $(0 \leq x \leq 1)$ and k is small.

In [8], parameter settings of $k = 10$ and $x = 0.2$ were shown to provide consistently good results across a variety of large and small collections studied. On average, EDLSI improved retrieval performance an average of 12% over vector-space retrieval. All collections showed significant improvements, ranging from 8% to 19%. Significant improvements over LSI were also noted in most cases. LSI outperformed EDLSI for $k = 10$ and $x = 0.2$ on only two small datasets, MED and CRAN. It is well known that LSI happens to perform particularly well on these datasets. Optimizing k and x for these specific datasets restored the outperformance of EDLSI.

Furthermore, computation of only a few singular values and their associated singular vectors has a significantly reduced cost when compared to the usual 100–300 dimensions required for traditional LSI. EDLSI also requires minimal extra memory during query run time when compared to vector-space retrieval and much less memory than LSI [8].

2.5 Sparse Column-Row Approximation

The performance of EDLSI suggested to us that the PSVD approximation that forms the basis for LSI may not be the best approximation for use in the context of information retrieval. EDLSI provides improvement to LSI results using a weighted combination of LSI and vector-space, and the best results are seen when the weighting of LSI is small, e.g., $x = 0.1$ or 0.2. However, the vast proportion of the work done in EDLSI in the computation of the LSI component.

The PSVD approximation used for LSI is well known to be the optimal rank-k approximation to A, such that the difference

$$\|A_k - A\| = \|U_k \Sigma_k V_k^T - U \Sigma V^T\|,$$

where $\|.\|$ can be the Frobenius norm or the 2-norm, is minimized. This does not necessarily correspond to optimality for information retrieval. A_k is also, in general, not sparse, even if A is. This leads to increased storage needs.

Stewart proposed in [14] a matrix approximation called the Sparse Column-Row Approximation (SCRA) based on a quasi-Gram Schmidt method for computing truncated, pivoted QR approximations for sparse matrices. The SCRA factors A into the product of three matrices: $Y \in \Re^{t \times s}, T \in \Re^{s \times s}$, and $Z \in \Re^{d \times s}$, where s is the reduced dimension. Then A is approximated as:

$$A \approx Y T Z^T.$$

The matrices Y and Z are formed by taking optimal QR approximations of A and A^T respectively, such that Y is composed of columns of A and Z^T is composed of rows of A. The matrix T is a dense matrix chosen such that, given Y and Z, the difference

$$\|Y T Z^T - A\|_F$$

is minimized.

The code accepts either a dimension s, or a tolerance τ that determines s. The matrices Y and Z are built up column by column, incorporating at each step the most important column of A or A^T, respectively. When the norm of the discarded portion of the matrix is smaller than the tolerance, or the reduced dimension s is reached, the factorization is finished.
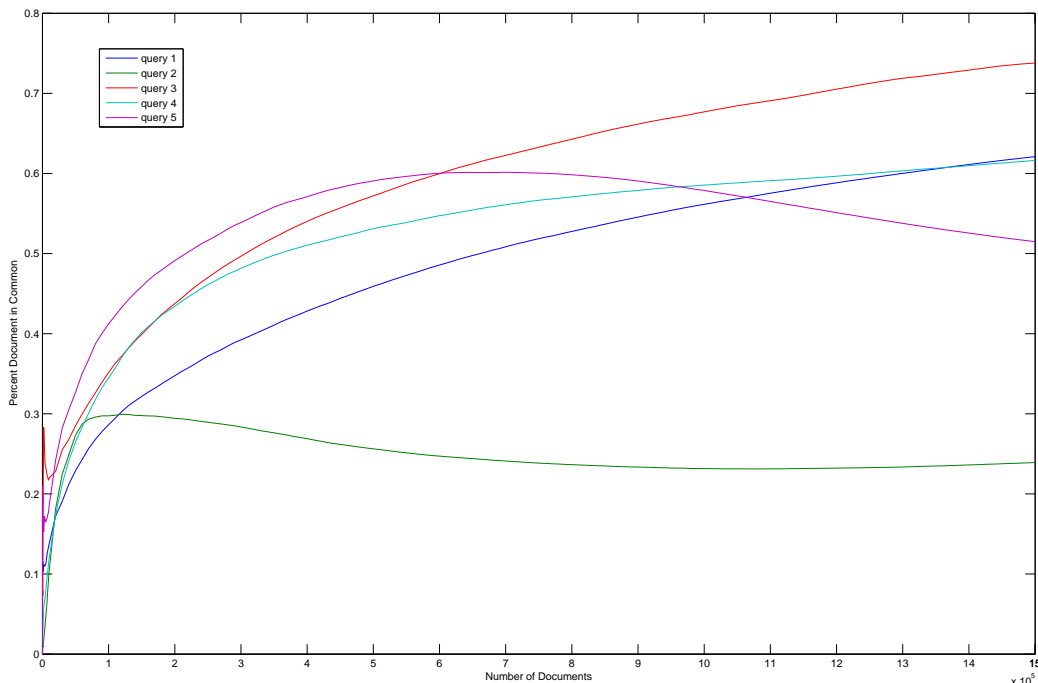
3 APPROACH

Our indexing of the dataset resulted in a term-document matrix of $486\,654$ terms and $6\,827\,940$ documents. We divided this into 81 pieces of approximately $84\,000$ documents each for ease in loading and working with the matrix.

Optimization runs were done for each of the three runs, using the set of document judgements given. These judgements resulted in a term-document matrix of $486\,654$ terms and $20\,090$ documents. When forming the matrix of relevance judgements, documents assigned relevance ratings of $0, -1$, and -2, as well as documents that were not judged for the query in question, were all assigned as non-relevant. Ratings of highly relevant and relevant were assigned as given. The optimization runs were performed for a range of parameters, and the parameters that gave the highest 11-point average precision, rounded to 2 decimal places, for all relevant documents were chosen as optimal, and were used for the full runs.

3.1 LSI with Folding-In

The first run we submitted used LSI with the folding-in algorithm for new documents. For the optimization run, the judged dataset was divided into an intial dataset of $10\,000$ documents, with increments of 500 documents each. The PSVD decomposition was performed on the initial set, and each increment was then folded-in. The run tested $k = 100$ to 1000 in steps of 100 at first, then

Fig. 4. Percent documents in common for queries 1–5

from 200 to 400 in steps of 10. The optimal value of k was 220, with an 11 point average precision of 0.19 for all relevant documents and 0.09 for highly relevant documents.

For the full run, the same approach was taken. A PSVD was performed on the first of the 81 pieces of the full set, then the remaining 80 were folded-in.

3.2 Distributed EDLSI

This run treated each division of the matrix as a separate collection, and performed EDLSI on each. The optimization run divided the judged dataset into 80 pieces, and after EDLSI was perfomed on each, the scores were collected and ranked overall. We intended to test values of k from 5 to 100 in increments of 5, and values of x from 0.1 to 0.5 in increments of 0.1. However, after seeing results up to $k = 45$, with the results steadily worsening, and being short on time, we stopped there. The optimal combination was $k = 5$ and $x = 0.1$, with an 11 point average precision of 0.20 for all relevant documents and 0.10 for highly relevant documents.

The full run proceeded similarly, with the 81 pieces of the matrix treated individually, then the results combined.

3.3 Distributed SCRA-Based LSI

Our final run made use of the SCRA to approximate the term-document matrix instead of the SVD. The run was fully distributed, with LSI performed on each piece, but using the SCRA in place of the PSVD. To optimize, the judged matrix was divided into 80 pieces. Instead of inputing a dimension s, we used a tolerance, which we calculated relative to the norm of the full matrix A. We tested tolerance parameters of 0.05 to 0.3 in steps of 0.05, and the absolute tolerance was computed by multiplying the relative tolerance with the norm of A. The optimal value for the tolerance was 0.10, with an 11 point average precision of 0.22 for all relevant documents and 0.12 for highly relevant documents.

The full dataset was treated similarly. Unfortunately, with 81 partitions and a tolerance of 0.10, the run would not have completed in time for submission. For this reason, the 81 pieces were each further subdivided into 40 partitions, and the tolerance was increased to 0.50. Though we did not test this tolerance, we note that the 11 point average precision for all relevant documents for the highest tested tolerance of 0.30 was only reduced to 0.17. This run was able to finish in time.

5

Fig. 5. Percent documents in common for queries 6–10
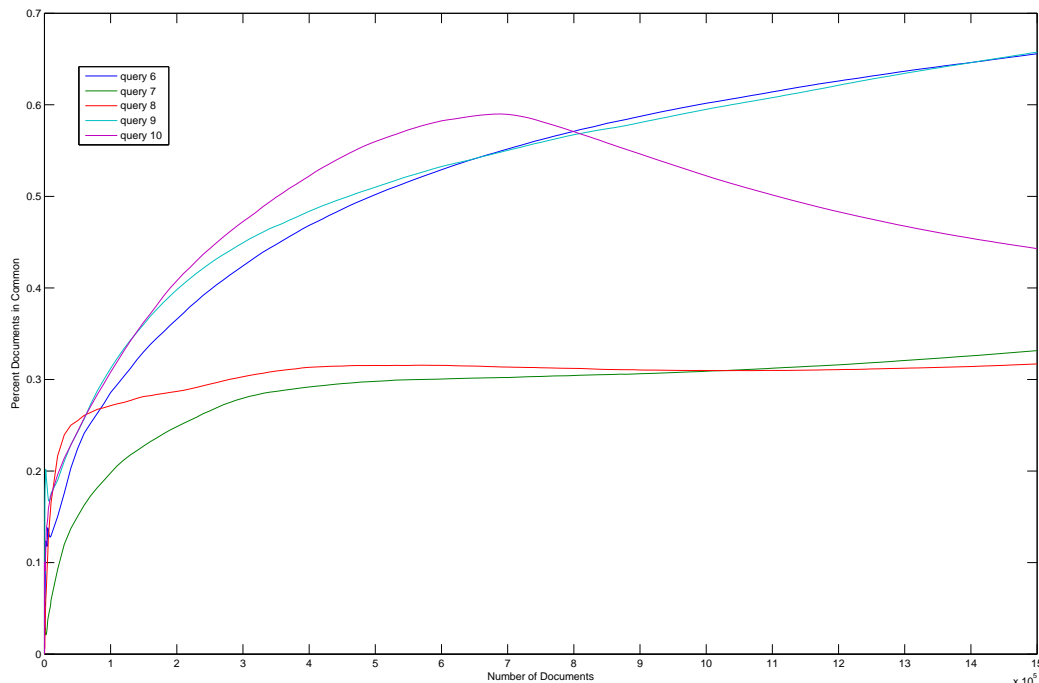
3.4 Setting K and K_h

Teams were required to set values of K and K_h for each query, indicating where they believe a balance between precision and recall is achieved. Table 1 shows the values we chose for each query. These values were the same for the three runs.

To set the value of K for each query, we decided to make use of the three runs. The runs used three different algorithms, and any document that was highly ranked by all three should be relevant. With this idea in mind, we calculated for each query the proportion of documents that all three methods had in common when looking only at the first n documents, where n is less than the maximum of $1\,500\,000$ documents that may be submitted. We then chose K to maximize percent documents in common. Figures 4 and 5 show the percent documents in common for queries 1–5 and 6–10 respectively.

To set K_h, we looked at the scores for each query and run. We noticed that for a small number of documents, scores were quite high, but the scores dropped off sharply within the first $1\,000$ documents. These high scores indicate that the documents are very 'close' in space to the query, and may indicate a higher relevance.

Accordingly, we chose K_h representing approximately the point where the scores had dropped off.

4 RESULTS

Figure 6 presents the results for our standard runs. At first glance the chart implies that LSI outperforms EDLSI and SCRA on the competition metric of Estimated F_1 at K. However, when the results are broken down by query, we see that all the performance on individual queries is split almost evenly between the three methods (see Table 7, which has the best performing method highlighted in each row). These results imply, to our surprise, that there is no clear advantage to any particular method.

In general the highly relevant results were not very strong, most likely due to a poor choice for the K_h value, which we continue to find very difficult to set (see Figure 8).

Interestingly, some queries continue to be much more difficult than others. Our results range from a high of 43.21% to a low of .41% (estimated F_1 at K), and we see many instances where estimated recall at K is reasonable (greater than 50%) but precision is so low as to make the

TABLE 1
K and K_h values for each query

Query number	K value	K_h value
7	1 500 000	500
51	100 000	300
80	1 500 000	600
89	1 500 000	100
102	600 000	1 000
103	1 500 000	500
104	300 000	300
105	400 000	500
138	1 500 000	1 000
145	700 000	1 000

Fig. 6. Result Summary
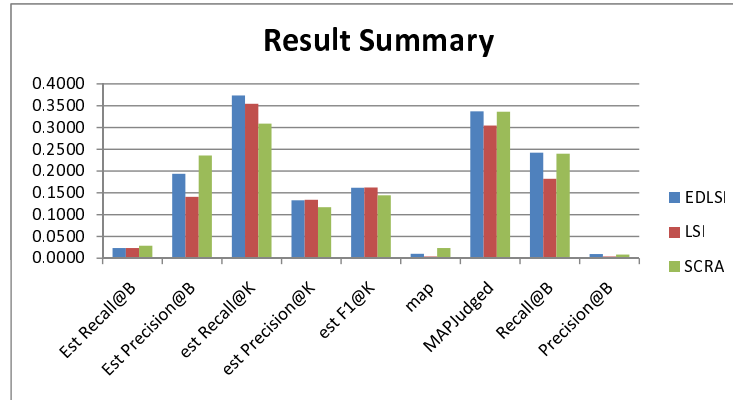
task unreasonable (finding 3% of 1,500,000 documents). We believe more research on determining appropriate K values is necessary. It appears as if retrieving large numbers of documents continues to the best approach for maximizing the competition metric.

5 CONCLUSION

We tested three LSI variants on a series of 10 queries on the TREC Legal batch corpus, the IIT Complex Document Information Processing (IIT CDIP) test collection, which contains approximately 7 million documents (57 GB of uncompressed text). Our early results show that all three methods have similar performance. However, a suboptimal parameter size was used for the SCRA method, due to time constraints. Additional experimentation with this method is warranted to determine if it outperforms EDLSI and LSI with folding-in.

The current three methods outperformed our LSI methods from 2008. Most of the gains appear to be obtained by selecting much larger K values; however,

precision at B (the number of documents retrieved by a boolean query run), which does not depend on K, is much higher this year.

REFERENCES

[1] R. A. Baeza-Yates, R. Baeza-Yates, and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley Longman Publishing Co., Inc., 1999.

[2] M. W. Berry, S. T. Dumais, and G. W. O'Brien. Using linear algebra for intelligent information retrieval. *SIAM Review*, 37(4):575–595, 1995.

[3] S. C. Deerwester, S. T. Dumais, T. K. Landauer, G. W. Furnas, and R. A. Harshman. Indexing by Latent Semantic Analysis. *Journal of the American Society of Information Science*, 41(6):391–407, 1990.

[4] S. T. Dumais. LSI meets TREC: A status report. In D. Harmon, editor, *The First Text REtrieval Conference (TREC1), National Institute of Standards and Technology Special Publication 500-207*, pages 137–152, 1992.

[5] S. T. Dumais. Latent semantic indexing (LSI) and TREC-2. In D. Harmon, editor, *The Second Text REtrieval Conference (TREC2), National Institute of Standards and Technology Special Publication 500-215*, pages 105–116, 1993.

[6] Glasgow Test Collections. http://www.dcs.gla.ac.uk/idom/ir_resources/test_collections/, 2009.

Fig. 7. Result Summary by Query

Result Summary by Query

Query	K	EDLSI			LSI			SCRA		
		Est R at K	Est P at K	Est F1 at K	Est R at K	Est P at K	Est F1 at K	Est R at K	Est P at K	Est F1 at K
7	1500000	0.4119	0.0532	0.0942	0.4650	0.0575	0.1024	0.5011	0.0645	0.1143
51	100000	0.0077	0.0020	0.0032	0.0036	0.0010	0.0015	0.0088	0.0027	0.0041
80	1500000	0.2216	0.0351	0.0606	0.2217	0.0362	0.0622	0.1852	0.0296	0.0511
89	1500000	0.6595	0.0093	0.0184	0.4247	0.0060	0.0119	0.2887	0.0041	0.0081
102	600000	0.4120	0.3554	0.3816	0.4104	0.3413	0.3726	0.3457	0.2888	0.3147
103	1500000	0.3238	0.2444	0.2785	0.3431	0.2440	0.2852	0.3854	0.2584	0.3094
104	300000	0.0824	0.1266	0.0998	0.0564	0.0920	0.0699	0.0538	0.0848	0.0658
105	400000	0.4752	0.1210	0.1929	0.5188	0.1368	0.2166	0.3962	0.1023	0.1626
138	1500000	0.6249	0.0412	0.0772	0.5675	0.0360	0.0677	0.5060	0.0352	0.0657
145	700000	0.5155	0.3393	0.4092	0.5323	0.3637	0.4321	0.4189	0.2995	0.3493
All	960000	0.3735	0.1328	0.1616	0.3544	0.1314	0.1622	0.3090	0.1170	0.1445

Fig. 8. Result Summary for Highly Relevant

Comparison of Performance for Highly Relevant
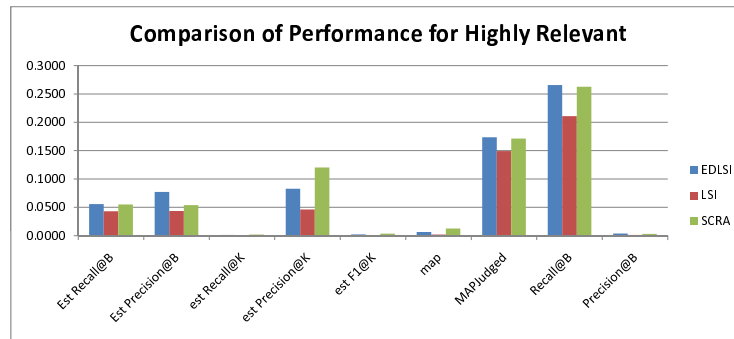
[7] E. R. Jessup and J. H. Martin. Taking a new look at the latent semantic analysis approach to information retrieval. In *Computational Information Retrieval*, pages 121–144, Philadelphia, PA, USA, 2001. Society for Industrial and Applied Mathematics.

[8] A. Kontostathis. Essential dimensions of latent semantic indexing (lsi). *Proceedings of the 40th Hawaii International Conference on System Sciences - 2007*, 2007.

[9] A. Kontostathis, W. M. Pottenger, and B. D. Davison. Identification of critical values in latent semantic indexing. In T.Y. Lin, S. Ohsuga, C. Liau, X. Hu, and S. Tsumoto, editors, *Foundations of Data Mining and Knowledge Discovery*, pages 333–346. Spring-Verlag, 2005.

[10] T. A. Letsche and M. W. Berry. Large-scale information retrieval with Latent Semantic Indexing. *Information Sciences*, 100(1-4):105–137, 1997.

[11] G. W. O'Brien. Information tools for updating an svd-encoded indexing scheme. Master's thesis, University of Tennessee, 1994.

[12] G. Salton and C. Buckley. Term-weighting approaches in automatic text retrieval. *Inf. Process Manage.*, 24(5):513–523, 1988.

[13] SMART. ftp://ftp.cs.cornell.edu/pub/smart/, 2009.

[14] G. W. Stewart. Four algorithms for the efficient computation of truncated pivoted qr approximations to a sparse matrix. *Numerische Mathematik*, 83(2):313–323, 1999.

[15] J. E. Tougas and R. J. Spiteri. Updating the partial singular value decomposition in latent semantic indexing. *ScienceDirect, Computational Statistics & Data Analysis*, 52:174–183, 2006.

[16] H. Zha and H. D. Simon. On updating problems in latent semantic indexing. *SIAM J. Sci. Comput., 21(2)*, pages 782–791, 1999.